"Advance human–machine interface automatic evaluation"

Gonzalez Calleros, Juan Manuel ; Guerrero Garcia, Josefina ; Vanderdonckt, Jean

Abstract

The need for accessibility evaluation tools is motivated by several endogenous and exogenous reasons coming from the end user (the designer and the developer) and companies releasing information systems. Existing evaluation tools mainly concentrate on examining the code of Web pages: Web pages more and more frequently contain non-HTML parts that entirely escape from being treated by existing techniques. This is the case of the advanced human–machine interface (AHMI), a piece of software programmed in C/C++, used for controlling the advanced flight management system in the aircraft cockpit. Studying this new user interface (UI) requires a structured approach to evaluate and validate AHMI designs. The goal in this work is to develop an evaluation tool to automate the process of evaluating the AHMI. The method addresses: support of multiple bases of guidelines (accessibility or usability or both) on-demand (partial or total evaluation), with different levels of details (a presentation...

Document type : *Article de périodique (Journal article)*

Advance Human Machine Interface Automatic Evaluation

Juan Manuel González Calleros[1], Josefina Guerrero García[1], Jean Vanderdonckt[2]

[1]*Benemérita Universidad Autónoma de Puebla, Computer Science Faculty, Puebla, Mexico*

E-mail: {jguerrero,juan.gonzalez}@cs.buap.mx – Web: http://www.cs.buap.mx/~jguerrero/

Tel: +52 222{229, 5500} – Fax: +52 222 {229, 5500}

[2]*Louvain School of Management, Louvain Interaction Lab, Université catholique de Louvain, Belgium*

E-mail : jean.vanderdocnkt@uclouvain.be

Abstract. The need for accessibility evaluation tools is motivated by several endogenous and exogenous reasons coming from the end user (the designer and the developer) and companies releasing Information Systems. Existing evaluation tools mainly concentrate on examining the code of Web pages: Web pages more and more frequently contain non-HTML parts that entirely escape from being treated by existing techniques. The goal in this work is to develop an evaluation tool that address: support of multiple bases of guidelines (accessibility or usability or both) on-demand (partial or total evaluation), with different levels of details (a presentation for a developers and a presentation for the person who is responsible for attributing the accessibility certification). For this purpose, an evaluation engine should be developed that perform guidelines evaluation or other independently of guidelines and usability knowledge. In addition, considering that right now, there is almost no consensus on how to uniformly evaluate guidelines automatically, a method is proposed to address this shortcoming.

1. INTRODUCTION

The largest part of automated accessibility and usability evaluation has been dedicated to examining the HTML code of a Web page to derive usability and accessibility problems and to repair them. For this purpose, information contained in HTML tags and their values are exploited to detect accessibility and usability flaws, defects by various techniques like static analysis, guidelines review. Separately, HTML only provides partial, yet insufficient, information to fully derive all information required for usability evaluation (Abrahão et al., 2008).

Other techniques exploit log files of users interacting with a system to increase the capabilities of examining the usability of it. WebRemUsine [Lecerof and Paternò 1998] does a good job in linking log files to a task model supported by the Web site to identify deviations between what the user actually does with the Web site and what he should do as prescribed by the task. For her very complete and extensive review, Ivory [2001] points out that techniques for automating the evaluation of Web sites, or at least for supporting the people in conducting evaluation and repair, are largely underexplored and seem to be promising. From her review [Ivory et al. 2003] and some analysis of the current state of the art, it seems to us that existing tools suffer from shortcomings, below they are described and a solution is proposed.

Right now, there is almost no consensus on how to uniformly evaluate guidelines automatically. For example, the two leaders of the market, Bobby and A-Prompt, are working

together right now to check that, when they automatically evaluate the WAI guidelines [Chisholm et al. 1999], they will reach the same conclusion. This is not the case today as both tools may produce different results for the same WAI guidelines since they are based on different interpretations of the WAI guidelines on the HTML code. This is very confusing to people who are responsible for designing and evaluating Web sites. The goal of the present work is therefore to develop a format of accessibility and usability guidelines that is common to all stakeholders so as to facilitate the communication of guidelines and their evolution over time, to guarantee that a same guideline will be understood, interpreted, and evaluated in the same way by all stakeholders.

Evaluation tools are mostly hard-coded: the most frequently used tool for evaluating accessibility guidelines, i.e. Bobby [Cooper 1999], is very good in identifying deviations from the WAI guidelines and guidelines from the Section 508. The common major shortcoming of the above existing evaluation tools is that the evaluation logic is hard coded in the evaluation engine, which makes them very inflexible for any modification of the evaluation logic or any introduction of new guidelines. In addition, many of them do not offer much possibilities of controlling the evaluation process like choosing which guideline to evaluate or the level of evaluation at evaluation time. The goal here is to develop an evaluation tool that addresses the above shortcomings, such as the support of multiple bases of guidelines (accessibility or usability or both) on-demand (partial or total evaluation), with different levels of details (a presentation for a developers and a presentation for the person who is responsible for attributing the accessibility certification). For this purpose, an evaluation engine should be developed that perform guidelines evaluation or other independently of guidelines and usability knowledge.

2. State of the art

The usage of accessibility evaluation tools always rely on the premises that accessibility guidelines could be turned into conditions to be verified on the input file. First of all, the process on transforming an accessibility guideline into an operational condition induces several restrictions during the various stages of this process. Second, simply stating an accessibility guideline does not mean that it automatically becomes an operational condition that could be checked immediately and straightforwardly on a Web site. Independently of the capability of an accessibility guideline to become operational, each evaluation tool also has some evaluation capability.

Farenc et al. [1996] reported that automatic evaluation of usability guidelines on windows-based application could reach a threshold of 44% of considered guidelines. Cooper et al. [1999] concluded that automatic evaluation of accessibility guidelines (in this case, W3C V1.0) could increase up to 50% of considered guidelines. It is expected that this plateau will be outreached in the near future. Not only because the guidelines are expressed in a more precise way, thus leading to an unambiguous interpretation of their application and their assessment but also because the evaluation tools exhibit more capabilities for evaluating more aspects that merely simple guidelines.

One of the possible avenues in the future is to combine analytical evaluation based on guidelines evaluation with empirical evaluation based on usage data. For instance, false negatives could be avoided by counterbalancing the false detections by usage data which contradict the analytical data. Conversely, the usage data could determine locations of the information system where further evaluation is desired or where no accessibility defects have been discovered. When one of the methods fails, the other one could compensate and enable identifying portions of the information system which should be subject to an in-depth analysis, manual or automated. The combination of a static analysis together with a dynamic analysis has a lot of promises that yet remain to be discovered.

Extraction of useful information from information systems to check it against accessibility and usability can be accomplished in several ways, including manual examination of code and automated static analysis. Static analysis of Information systems examines the code without interpreting or executing it in order to understand aspects of the information system [Beirekdar 2002], static analysis has been successfully used in software testing and compiler optimization. There are other techniques to automatically evaluate information systems such as the following methods:

- Syntactic Analysis and Grouping [Van Sickle et al. 1993] relies on a recognition algorithm that identifies input/output statements and attempts to incorporate them into groups. The grouping information is then used to define screens from the original user interface. This is particularly appropriate for scripting languages.

- Cliché and Plan recognition [Wills 1990] automatically identify occurrences of clichés, stereotyped code fragments for algorithms and data structures. The cliché recognition system translates the original code into a plan calculus, which is then encoded into a flow graph, producing a language-independent representation of the interpretation's flow that neutralizes syntactic variations in the code.
- Slicing techniques [Gallagher and Lyle 1991] examine selected, related portions (slices) of code to analyze behavior and to recover design decisions. The code statements in a program slice are not necessarily contiguous; this facilitates design recovery even if the code is poorly structured. This seems promising for SVG code.
- Abstract interpretation [Lecharlier 1996] parses the entire code to build an abstract representation of it that can be virtually executed based on first-order predicated expressing conditions before and after each instruction. This technique seems particularly appropriate for examining Java and JavaScript.

All above methods suffer from one common drawback: they work without taking into account the interaction with the end user. Thus, arises the need to consider advanced techniques that combine both static part (without users) and dynamic part (with users). These techniques are grouped under the umbrella of dynamic analysis, where the information system are analyzed during execution and interaction with users), a promising avenue to pursue for further research on this problem. Dynamic analysis [Ritsch and Sneed 1993] automatically instruments the code by a dynamic analyzer. A test monitor captures output from the instrumentation during program execution. The output is then analyzed to determine program structure and components.

3. Automated evaluation: The Advanced Human Machine Interface Case Study

The AFMS is a piece of software that helps pilots to manage their flight in terms of trajectory production (e.g. generate trajectories out of a constraint list) [Mollwitz 2006]. The AFMS can be handled via a new system called Advanced Human Machine Interface (AHMI) by pilots. The interaction between the pilot and the AHMI is through the different User Interfaces that composed the AHMI. The AHMI is composed of traditional control objects (buttons, spin button, menu) and non-traditional (compass rose, aircraft). The transformation of the existing character-based UI for the AFMS (left hand side in Fig. 1) into a graphical User Interface (UI) (middle in Fig. 1) encounters new challenges for the development process (analysis, design, implementation, evaluation) and their future usage. At least for two reasons: evaluation of this UI is costly (in terms of assets and their availability) and the design must be rigorous.

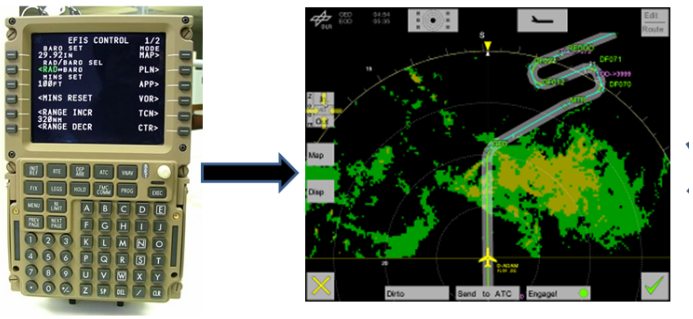

Figure 1 AFMS evolution

3.1 Modeling the Advanced Human Machine User Interface

The evaluation of the AHMI considers static aspects (UI layout, position of objects) and dynamic concepts (state of a button during the interaction, color of the label). Extracting this information directly from the code defining the AHMI is feasible but would be hard to obtain.

Expressing the UI with models would benefit this task because: static and dynamic aspects would be stored in the models and then the models would be object of further evaluation to check, for instance, usability guidelines over the UI objects (distribution of the widgets composing the UI).

A formal underpinning for describing models in a set of meta-models facilitates meaningful integration and evaluation of such models, and is the basis for automation through software. Models are represented as a UML class diagram that then can be specified using the UsiXML language (Limbourg et al. 2005) and related software, e.g., GrafiXML (Michotte and Vanderdonckt, 2008) and FlowiXML (Guerrero et al. 2008). UsiXML stands for USer Interface eXtensible Markup Language. It is a User Interface Description Language (UIDL) that consists of a high-level computer language for describing characteristics of interest of a UI with respect to the rest of an interactive application (Bodart et al., 1995). UsiXML involves the definition of a syntax (i.e. how these characteristics can be expressed in terms of the language) and semantics (i.e., what do these characteristics mean in the real world). It can be considered as a common way to specify a UI independently of any target language (e.g., c++ and openGL for the AHMI) that would serve to implement this UI.

UsiXML Concrete User Interface Model (CUI) allows both the specification of the presentation and the behavior of an AHMI with elements that can be perceived by users (Vanderdonckt 2006). The CUI model as an abstraction of AHMI elements includes objects in the UI that are different from those found in traditional toolkits (widgets set found in popular toolkits, e.g., Java AWT/Swing, HTML 4.0, Flash DRK 6).

The description of the models is just to give the lector the taste of the problem without being exhaustive and too detailed. The resulting model is depicted in Figure 2. A map that groups aircrafts, airports, trajectories, navigation aids. It has several attributes such as: view Mode (head-up or north-up), flags to determine whether the different elements on the map are visible or not. The vertical profile is the vertical view on the Navigation Display. It has attributes such as: distance scale (for instance, nautical miles), distance Steps to define steps between the beginning and the end of the distance, time scale (the format of the time), time steps to define steps between the beginning and the end of the distance, cruise flight level, the intercept altitude at which the localizer is intercepted. The vertical profile is composed as well by two other objects: speed profile that represents the speed for the trajectory of the aircraft and the altitude profile corresponding to the altitude scale at the left edge, which shows hundreds of feet. Other models were created to represent the bird view of the aircraft including the compass rose, the track that the aircraft is flying.

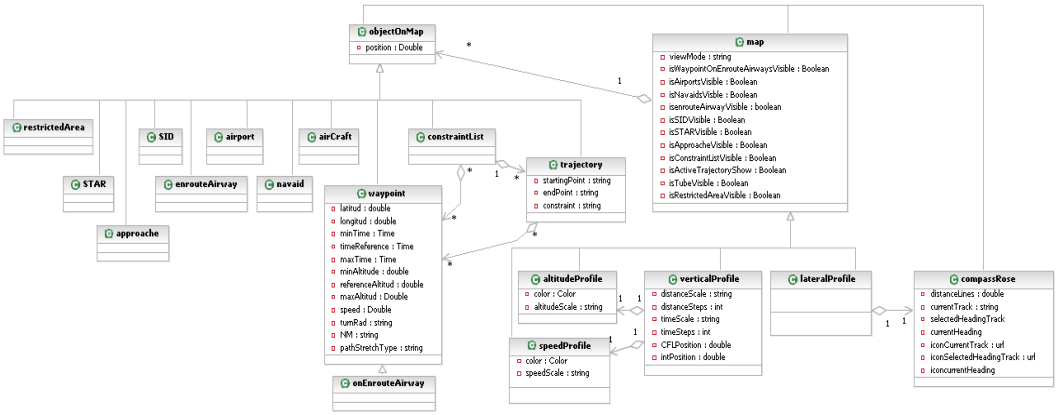

Figure 2. Graphical Individual Components used in the AHMI

The objective of the case study is to describe the automatic evaluation of the AHMI in the context of a setup called Virtual Simulation Platform (VSP) that simulates the AHMI system with a symbolic AHMI (SAHMI) in order to be able to predict pilot's behavior; pilots are substituted with a cognitive model. The SAHMI is related to a cognitive architecture (CA) that is in charge of simulating pilot's behavior. There is a constant communication between these two pieces of software as the CA interacts with the SAHMI and not with the real system. The tool chain, including the CA, is depicted in Fig. 3. The task of the SAHIM is to monitor and send messages, related to AHMI interaction, to the CA. This means that to execute an action over the AHMI the CA sends a message to the data pool indicating the triggered event (e.g. click on negotiation button). Then, the SAHMI reads the message from the Datapool and analyze its content in order to provide a feedback to the CA. Based on the action to perform, the status of the CHIME and the

status of the Datapool the set of action is triggered internally (e.g. change state of the system to indicate negotiation) to update the CHIME and Datapool. Finally the feedback, a set of messages with information related to the visual feedback resulting from this action, is sent to the Datapool. The messages are accessed by the CA. The lector can find more details on the CA in (Lüdtke et al. 2009).

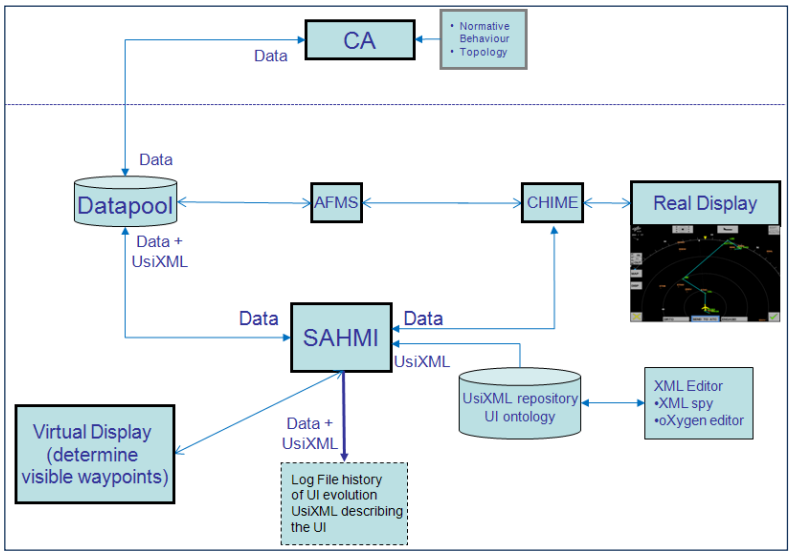

Figure 1 Tool Chain of the Symbolic AHMI

3.2 Automated Evaluation Process

We have seen that to wire the knowledge based on ergonomic rules and recommendations procedure (s) is a major challenge in existing tools. To address this shortcoming, a technique is proposed that separates the knowledge base of evaluation engine automatically and perform evaluation over dynamic and static aspects of the UI. Fig. 4 illustrates the global process for automatic evaluation. The knowledge base of ergonomic rules and accessibility is collected to assist the application. This knowledge base is a collection from ergonomic guidelines, structures (Smith and Mosier 1984) or various recommendations (WCAG 2.0, AccessiWeb) that are encoded in a formal format using the UsiXML language. The knowledge base, stored in a text file, is used by the UsabilityAdviser parser to load ergonomic rules in the computer memory in data structure. Once this internal structure is created the tool performs a data analysis of the User Interface (UI), encoded in a compatible format (UsiXML) developed in a UsiXML editor. The UsabilityAdviser search for violations of rules formalized through the automatic evaluation of UI data. Finally, a report on the errors of ergonomics and accessibility is published.

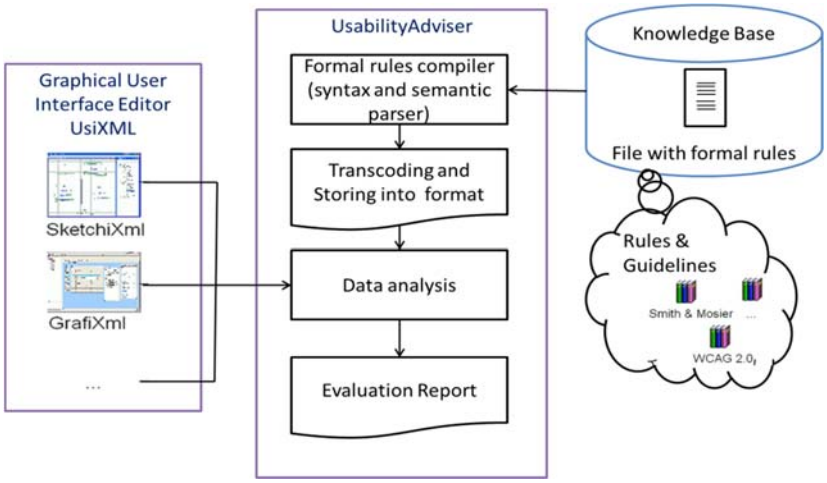

Figure 4. Global process for automatic evaluation

Ergonomic rules and recommendations are normally expressed in a textual format. So, they have to be encoded according to a well-defined syntax. Once rules are encoded they are checked with lexical and syntactic parser to determine error in the encoding of rules. Once the lexical and syntactic analysis is completed, the formal rules will be considered compliant and ready to be used for UI evaluation. UI evaluation is done directly on a UI editor or to a file containing a UI description.

A complete review and compilation of existing rules from different sources was done to create and update the knowledge base of ergonomic rules. The process of collecting rules several attributes can be used to describe a guideline. This allows users to be aware of the fact that its ergonomic error has to be taken into account, should be taken into account or could possibly be taken into account. Indeed, some rules are important and should absolutely be corrected, while others are less important and produce just warnings:

- Category referring to the group that gathers a common characteristic of the guideline, such as: visibility, information density, consistency, etc.
- Importance defines the level of importance of each guideline. Each guideline can be annotated with an indication on their importance. The MoSCoW list is often used to indicate the importance. MoSCoW stands for: MUST have this, when several sources mention the rule; SHOULD have this if at all possible, some sources mention the rule; COULD have this if it does not affect anything else, few sources mention the rule; and WOULD like in the future, just one source mention the rule but the source is a strong one.
- Evidence denoting the source of the guideline and its validity. Ranking from: low, medium, high. To be present in this list at least one reference must exist to this guideline (low). If there is empirical study that proves the guideline then is medium. If two or more studies get the same results then the evidence is high.
- Validation type denoting the way a guideline can be evaluated: manually, automatically or a combination of both.
- Citation, list the sources from which the guideline was retrieved.
- Example, describes an example from the literature and its source come along with the guideline.
- Users, list the categories of users that are affected by this rule: regular user, visual impaired.
- Platform, list the device and operating system that can be used for this rule
- Environment, list the possible abstract places where the rule can be applied: hands-off, eyes-off.

The interpretation of the rule in natural language depends greatly on the level of abstraction of the rule and the understanding of the user. When ergonomic rules were properly interpreted, it only remains to translate these interpretations of natural language in a formal language. No fault of syntax can appear in the formalization of rules. As guidelines are means to provide designers with guidance on to accomplish their goals in a systematic way. Designs can be cross checked against the guidelines in different ways:

- (M)anually. A guideline related to a UI characteristic that cannot be automatically evaluated. For instance, objects should be similar as much as the real objects.
- (A)utomatically. A guideline referring to characteristics that could be automatically evaluated if the design and the guideline can be expressed in a machine readable format. For instance, check the correct color combination to avoid accessibility problems.
- (P)artially automated. A guideline that can be partially automatically evaluated. For instance, non-text content should be described with a text-alternative describing the object. While the presence of text can be checked the accuracy of its description can be just manually evaluated.

Once the rules are formalized, the UsabilityAdviser verifies its compliance. Each rule consists of a header containing a formal description, composed of a set of *variables* and *constraints*. The header is a formal description that is interpreted by a computer program to test rules in a UI. Constraints, in turn, express the semantic rules given to describe the properties and to respect for the rule to be verified.

Variables declarations must be separated by a comma. The declaration of a variable considers: a quantifier indicating the number of elements affected by this variable; a widget indicating the actual widgets affected by this variable; an identifier; and new identifier, to avoid repetition of

variables we need to explicitly mention that there are no repetition. The list of quantifiers we use to write variable declarations is: *all* referring to universal rule, that is to say either "For everything ..."; *one or more* referring to the rules of existential type, that is to say either "it exists ... "; *at least n* referring to rules like "At least n widgets ..."; *at most n* referring to rules like, "No more than n widgets ..."; *exactly n* referring to rules such as "Exactly ... n widgets"; *not n* referring to "should not have n ... widgets"; and *no* referring to "No widget".

Constraint defined in the ergonomic rules concerns at least one variable. Several constraints can be defined for each variable. If the final value of evaluating a single rule returns true, then the rule is respected; otherwise, it means that at least one constraint is not fully satisfied. Constraints play a central role in validating rules as they describe the properties to respect for the rule to be verified. For the sake of understanding and clarity, we will give an example of a complete rule for each type of constraint:

- *has_properties*: This constraint will test whether the attribute of a widget has the expected value by comparing it with a constant or an attribute of another widget. For instance, testing whether there are subtitles for each video (*all* videoComponent v: v *has_properties* (subtitle ==true)).
- *Contains*: this constraint allows expressing the fact that a widget must contain another widget in a specific amount. For instance, a rule limiting the number of allowed images in a toolbar (*no toolbar* t, *at_least* 20 imageComponent i : t *contains* (i)).
- *is_correctly_balanced*: This constraint ensures that widgets distribution is properly balanced. A Graphical User Interface (GUI) unbalanced is depicted in Figure 5. This rule will receive two integers as parameters designating a number of pixels. The number of pixels respectively defines the minimum and maximum space that must be respected between a widget and all the other widgets. The minimum and maximum space should be respected equally by the widget and the border of the window. To show an example of this constraint, we set the space between each widget range from 30 to 80 pixels (*all* graphicalIndividualComponents g: g *is_correctly_balanced* (30,80)).
- has_attribute_value_in_file: This constraint will test whether the attribute of a widget is contained in a file that is specified as a parameter. The content of this file is loaded into memory when compiling rules. We will formalize this constraint using a ruler to verify that a standard font is used (*all* outputText o: o *has_attribute_vale_in_file* (textFont, "policeStandard.txt")). The list of fonts is contained in the file "policesStandard.txt".
- has_external_constraint: This constraint will call a Java method defined by the user. The parameters required are the name of the class defined by the user, followed by a dot, then the name of the method (within the class of the user) to call with an open parenthesis and closing. For instance, *all* videoComponents v: v *has_external_constraint* (ExternalClass.getBool()). A Java method is defined in a separated file including a class "ExternalClass" and the method "getBool". This method must return an object of type Boolean. This is one of the must powefull features or our solution, so, we will come back to its peculiarities later in the next section on implementation.
- Linking constraints: there are three different types of links to associate constraints. The first type of link is the logical AND (symbolized by the keyword 'and' in the formal language). The second type of link is the logical OR (symbolized by the keyword 'or' in the formal language). The third type of link is an implication (symbolized by ->). One implication is composed of a premise and a consequent. This premise and the result are both composed of at least one constraint which can optionally be linked via 'and' and 'or'. The principle of involvement may be reflected by the following formula: "If the premise is true, then the consequent must be true". A simple example illustrates this: we want to verify that each comboBox that can be edited has a default text value describing its contents. We interpret this rule with the phrase: "If a ComboBox is editable, it must have a default text value (*all* comboBox c: c *has_properties* (isEditable = = true) - > c *has_properties* (defaultcontent != "")).

3.3 Advanced Machine User Interface Evaluation

The first step of this phase is to model the characteristics of each of the widgets using UsiXML. Widgets have properties and those properties are examined in order to validate the different ergonomic rules in the UI description file. To take advantage of each of these properties, all widgets were modeled. SAHMI widgets information is sent to the UsabilityAdviser. UsabilityAdviser "Knowledge Base" contains the set of formalized to check ergonomics and

accessibility. The knowledge base is used by the “Formal rules compiler” to load and parse the rules. Once this internal structure is created the tool performs a data analysis of the UI, encoded in UsiXML, which may be developed in a UsiXML editor. For instance, a rule to check perception and readability of menus in the SAHMI states “Avoid the use of sentences for labeling menu items”. Menus must include items named in a consistent manner, succinctly and accurately. The use of these can offer a quick selection to the user, and there is no question of introducing items with labels that are complete sentences, we define the following values for the rule properties:

- *Ergonomic Criteria*: Content Perception. Indeed, the use of menus should be simple and its perception of their contents quick.
- *Priority level*: Recommendation. Indeed, proper use of menus is strongly recommended.
- *Certainty*: Medium. As discussed in the interpretation of this rule, it is difficult to identify whether the wording of an item is a complete sentence or not.
- *Degree of coverage*: Partial, because it may be that certain phrases in the texts is not found, but nevertheless remain quite rare.

The interpretation of this rule addresses the UsiXML widgets entitled 'item' (is an abstraction that can be used to represent menu items, comboBox, listBox items). The rule is particularly needed for widgets 'Tree' and 'Listbox'. In UsiXML specification every widget has an attribute called 'defaultContent'. This attribute contains information about the wording of the item. To discover whether the wording of the item is a sentence or not, the number of whitespaces is counted. To our understanding the more white spaces, the more likely that this language is a sentence. We will set a value N which symbolizing the maximum number of white spaces. Beyond this limit N, we consider that the wording is a sentence. Our interpretation of this rule will be: "For each item, the wording of this item has at most N consecutive white space 'with N = 2 in our case`”. The following regular expression formalizes this rule:

```
header : 1, "Avoid the use of sentences for labelling menu items", medium,  partial,
            "Content Perception", should, enabled
rule : all item i : i has_properties (defaultContent !="\w{1,}\s\w{1,}\s.* ")
```

The UsabilityAdviser search for violations of rules formalized through the automatic evaluation of UI data. Finally, a report on the found violations of ergonomics and accessibility is presented.

4. Lesson Learnt with AHMI Automated Evaluation

Evaluation of the AFMS is hard due to its lack of semantic information about the actions performed over it considering that is just a physical device. Evaluations about pilot’s interaction require expensive setups and a lot of human manual work in the debriefing and then a final interpretation of the results that might include interviews again with the actors involved in the experiments. Also, the AHMI being just code requires a complex manipulation of the code to create scripts describing the UI evolution is the evaluation is desired. This solution could satisfy basic needs but if there is a changed on the UI then the solution no longer works. Modeling the SAHMI showed to be an option for UI evaluation. The model of the UI, as described in section 3.1, can be modified in order to test different UI configurations. The coupling of a virtual model of the cockpit to the real one with model-based approaches allows the analysis, checking, and validating cockpit design. Traditional measurements can be assessed like UI workload, color combination.

Adapting the UsabilityAdviser on the VSP, i.e., an evaluation layer of the SAHMI added the capacity to the system to trace the evolution of the UI during the interaction with the CA. Then, empirical test on the UI can be done including: time to achieve a task, success on performing task. Simulated pilots actions over the UI are passed as messages that are processed in the model merger. These data from the simulation system must be transformed to be compatible with UsiXML format. This data is store as a log File history that we called UI evolution over time as the SAHMI is (Fig. 5- A). By detecting UI ergonomic violations over each frame (UI) of the log file a possible solution is evaluated (Fig. 5- B) to determine whether a pilot would be faster or would have a reduced workload to perform his task. Notice that new UI configurations are unlimited, as many widgets can be used to perform a task. For instance, in Fig. 5- t3, a group of check boxes (left inferior corner in Fig. 5-B) replaces a set of toggle buttons (middle in Fig. 5-A). Even though, instead of check boxes other widgets could be used, such as: list box, combo box or

even the same group of toggle buttons with a different layout. The goal is to maximize performance and to reduce workload.

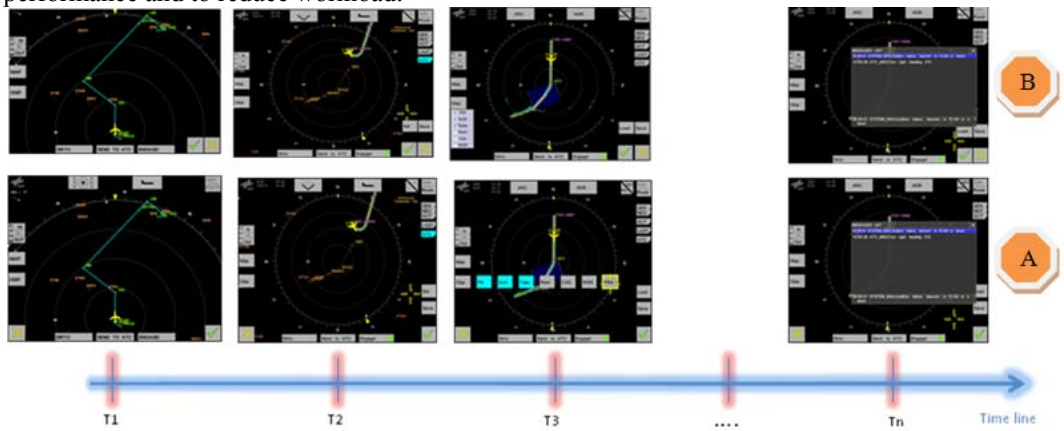

Figure 5. AHMI UI Evolution over time (A) and modified version over the same UI evolution (B)

The advantage of having a model is that starting from it further investigation is performed to identify the type of guidelines that are of interest and that are valid for checking. The drawback on this approach is that user's feedback is missing. There is no way to collect intentions and the subjective feeling of the user while interacting with the "new" system. Also, considering just static aspects of the AHMI is not enough for a complete evaluation. We need to consider the dynamicity of the AHMI that stress the need to consider those UI elements that are constantly changing, for instance, to identify in which context the engage button gets green. To evaluate those dynamic data from behavior model of the CA is needed for further evaluation.

5. Conclusion

After evaluating the system new requirements are established and their implementation might be costly, in terms of time, assets and technology involved. There are many techniques applied to evaluate user interfaces; some techniques can be applied during the development phase (e.g. heuristic analysis), while others have to be used only when the design or prototype is finished (e.g. formal analysis). Evaluation involves several activities which depend on the method used. The activities more often used are: Capture: it consists of collecting usability data, such as task completion time, errors, guideline violations, and subjective ratings; Analysis: is the phase in which usability data are interpreted to identify usability problems in the interface; Critique: consists of suggesting solutions or improvements to mitigate the problems identified. Many methods have been proposed because different techniques reveal different usability problems: usability evaluation typically only covers a subset of the possible actions users might take. Furthermore, different testers may detect different problems even if the same evaluation method is used. Thus, further efforts (e.g. increasing the evaluator team) are required. One solution to this issue is trying to automate all, or part of the evaluation process activities (capture, analysis, critique).

We proposed an approach for automated UI evaluation with a cognitive architecture, which is usable in industrial application. It uses a model driven approach, and is connectable to tools that are already in use in the industry, like Matlab or Scade, which allows re-use of the models defined by the system designers. UsiXML provides a model driven development for the industry. The AHMI is a new innovative system that introduces new challenges for the development of cockpit systems. Development steps including design and evaluation, among others, are normally limited addressed when it refers to the UI. Design knowledge is normally hidden and evaluation is mostly focused on the system functionality rather than of the usability of the system. Modeling the SAHMI showed to be an option for UI evaluation. The model of the UI, as described in this article, can be modified in order to test different UI configurations. These models together can be used for automated UI evaluation, for which the HUMAN project has already implemented some prototypical tools. Still an open issue is the connection between the cognitive architecture system (CASCaS) and UsiXML.

Acknowledgements. We gratefully acknowledge the support of the Human European project (Model based Analysis of Human Errors during Aircraft Cockpit System Design, project funded by FP7-AAT-2007-RTD-1/CP-FP-211988 from European Commission), the ITEA2 Call 3 UsiXML project under reference 20080026.

REFERENCES

Abrahão, S., Iborra, E., Vanderdonckt, J., Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool, in Law, E., Hvannberg, E., and Cockton, G. (eds.), "Maturing Usability: Quality in Software, Interaction and Value", Chapter 1, HCI Series, Vol. 10, Springer, London, 2008, pp. 3-32.

Beirekdar, A., Keita, M., Noirhomme, M., Randolet, F., Vanderdonckt, J., Mariage, C., Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites, Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005 (Rome, 12-16 September 2005), Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 281-294.

Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Vanderdonckt, J., Zucchinetti, G., Key Activities for a Development Methodology of Interactive Applications, Chapter 7, in Benyon, D., Palanque, Ph. (Eds.), "Critical Issues in User Interface Systems Engineering", Springer-Verlag, Berlin, 1995, pp. 109-134.

Chisholm, W., Vanderheiden, G., Jacobs, I., Web Content Accessibility Guidelines 1.0, W3C Recommendation, 5 May 1999, accessible at http://www.w3.org/TR/WAI-WEBCONTENT/

Cooper, M., Evaluating Accessibility and Usability of Web Sites, Proc. of 2nd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999), A. Puerta & J. Vanderdonckt (eds.), Kluwer Academics Publisher, Dordrecht, 1999, pp. 33-42.

Cooper, M., Limbourg, Q., Mariage, C., Vanderdonckt, J., Integrating Universal Design into a Global Approach for Managing Very Large Web Sites, Proc. of the 5th ERCIM Workshop on User Interfaces for All UI4ALL'99 (Dagstuhl, 28 November-1 December 1999), A. Kobsa & C. Stephanidis (eds.), GMD Report 74, GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, 1999, pp. 131-150, accessible at http://ui4all.ics.forth.gr/UI4ALL-99/Cooper.pdf

Farenc, Ch., Liberati, V., Barthet, M.-F., Automatic Ergonomic Evaluation: What are the Limits?, Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996), J. Vanderdonckt (ed.), Presses Universitaires de Namur, Namur, 1996, pp. 159-170.

Gallagher, K.B., Lyle, J.R., Using Program Slicing in Software Maintenance, IEEE Transactions on Software Engineering, Vol.17, No. 8, 1991, pp. 751-761.

Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., Muñoz-Arteaga, J., A Theoretical Survey of User Interface Description Languages: Preliminary Results, Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHC'2009 (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 36-43.

Guerrero, J., Vanderdonckt, J., Gonzalez, J., FlowiXML: a Step towards Designing Workflow Management Systems, Journal of Web Engineering, Vol. 4, No. 2, 2008, pp. 163-182.

Ivory, M.Y., Hearst, M.A., State of the Art in Automating Usability Evaluation of User Interfaces, ACM Computing Surveys, Vol. 33, No. 4, December 2001, pp. 470-516.

Ivory, M.Y., Mankoff, J., Le, A., Using Automated Tools to Improve Web Site Usage by Users with Diverse Abilities, IT&Society, Special Issue on Web Navigation Skills, Vol. 1, No. 3, 2003, pp. 195-236, accessible at http://www.stanford.edu/group/siqss/itandsociety/v01i03/ v01i03a11.pdf

Lecharlier, B., Abstract Interpretation and Application to Interactive System Verification, Proc. of 3rd Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'96 (Namur, 5-7 June 1996), F. Bodart & J. Vanderdonckt (eds.), Springer-Verlag, Vienna, 1996, pp. 46-72.

Lecerof, A., Paternò, F., Automatic Support for Usability Evaluation, IEEE Transactions on Software Engineering, Vol. 24, N°10, October 1998, pp. 863-888.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCIDSVIS'2004 (Hamburg, July 11-13, 2004). Springer-Verlag, Berlin (2005).

Lüdtke, A., Weber, L., Osterloh, J.P., Wortelen, B., Modeling Pilot and Driver Behavior for Human Error Simulation. HCI (11) 2009: 403-412.

Michotte, B., Vanderdonckt, J., GrafiXML, A Multi-Target User Interface Builder based on UsiXML, Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), D. Greenwood, M. Grottke, H. Lutfiyya, M. Popescu (eds.), IEEE Computer Society Press, Los Alamitos, 2008, pp. 15-22.

Mollwitz, V. AFMS Handbook for Users. Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). June 2006.

Ritsch, H., Sneed, H.M., Reverse Engineering Programs via Dynamic Analysis, Proc. of Working Conference on Reverse Engineering WCRE'93 (Baltimore, 21-23 May 1993), IEEE Computer Society Press, Los Alamitos, 1993, pp. 192-201.

Smith S.L. and Mosier J.N., Design guidelines for user-system interface software. Technical Report NTIS No. AD A 154 907. 1984.

Van Sickle, Larry, Liu, Zheng Yang, and Ballantyne, Michael, "Recovering User Interface Specifications for Porting Transaction Processing Applications", EDS Research, Austin Laboratory, 1601 Rio Grande, Suite 500, Austin TX 78701, 1993

Vanderdonckt, J., Beirekdar, A., Automated Web Evaluation by Guideline Review, Journal of Web Engineering, Vol. 4, No. 2, 2005, pp. 102-117.

Wills, Linda Mary. "Automated Program Recognition: A Feasibility Demonstration", Artificial Intelligence, Elsevier Science Publishers B.V., (North-Holland), 1990.